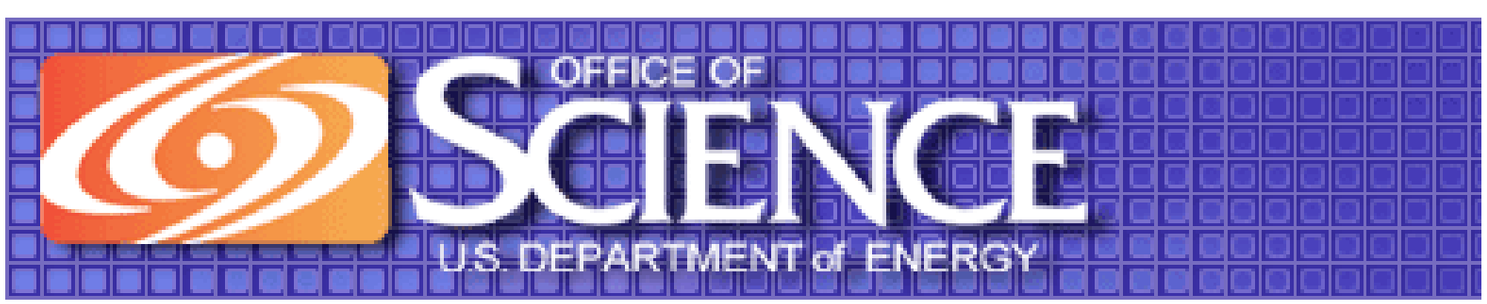
A Model Coupling Toolkit Primer

J. Walter Larson
Mathematics and Computer Science Division
Argonne National Laboratory

Presented at the Sixth DOE ACTS Collection Workshop
Berkeley, CA
August 23-36, 2005



Overview

- What is MCT?
- Multiphysics Models
- The Parallel Coupling Problem
- How MCT Enables Solutions to the Parallel Coupling Problem
- The MCT Programming Model
- An Illustrative Example
- Conclusions/Future

The MCT Infomercial...

Man: What are grits?
Waitress: They're fifty cents
Man: But what *are* they?
Waitress: They're extra.

-from New Jersey Turnpike, USA I-IV
by Laurie Anderson

What is MCT?

- MCT := *Model Coupling Toolkit*
- A software toolkit for coupling message-passing-parallel models into a single *parallel coupled model*
- An extension to MPI (yes, we are MPI-specific--*for now*)

Who Wrote MCT?

- According to Forrest Hoffman (ORNL) in a Linux Magazine column: "...fearless programmers from Argonne National Laboratory"
- J. Larson and R. Jacob--lead developers, along with E. Ong (UW Madison), J. Guo (NASA GSFC), J. Mogill and C. Corey (Cray), and R. Loy (ANL)

Why Was It Built?

- We needed it to build a fully message-passing parallel coupled climate model, the Community Climate System Model (CCSM)
- We felt this was an emerging problem of central importance in computational science
- Somebody was willing to pay for it

Who Paid for It?

- Department of Energy Office of Science
- Specifically the DOE Office of Biological and Environmental Research
- Through the Climate Change Prediction Program (CCPP), which is part of DOE's Scientific Discovery through Advanced Computing SciDAC program
- Funded 2000-present to support development of CCSM, which DOE uses to perform USA's climate change assessment studies

How Can I Get It?

- Open source--MCT is available for free and freely available from the MCT Web site

http://www.mcs.anl.gov/mct

- MCT is highly portable and will build and run on most unix systems with most compilers. This includes most commodity Linux clusters and also vector supercomputers such as the Cray X-1, NEC SX- series (including the Earth Simulator), and Fujitsu VPP.

MCT 2.1.0 Distribution

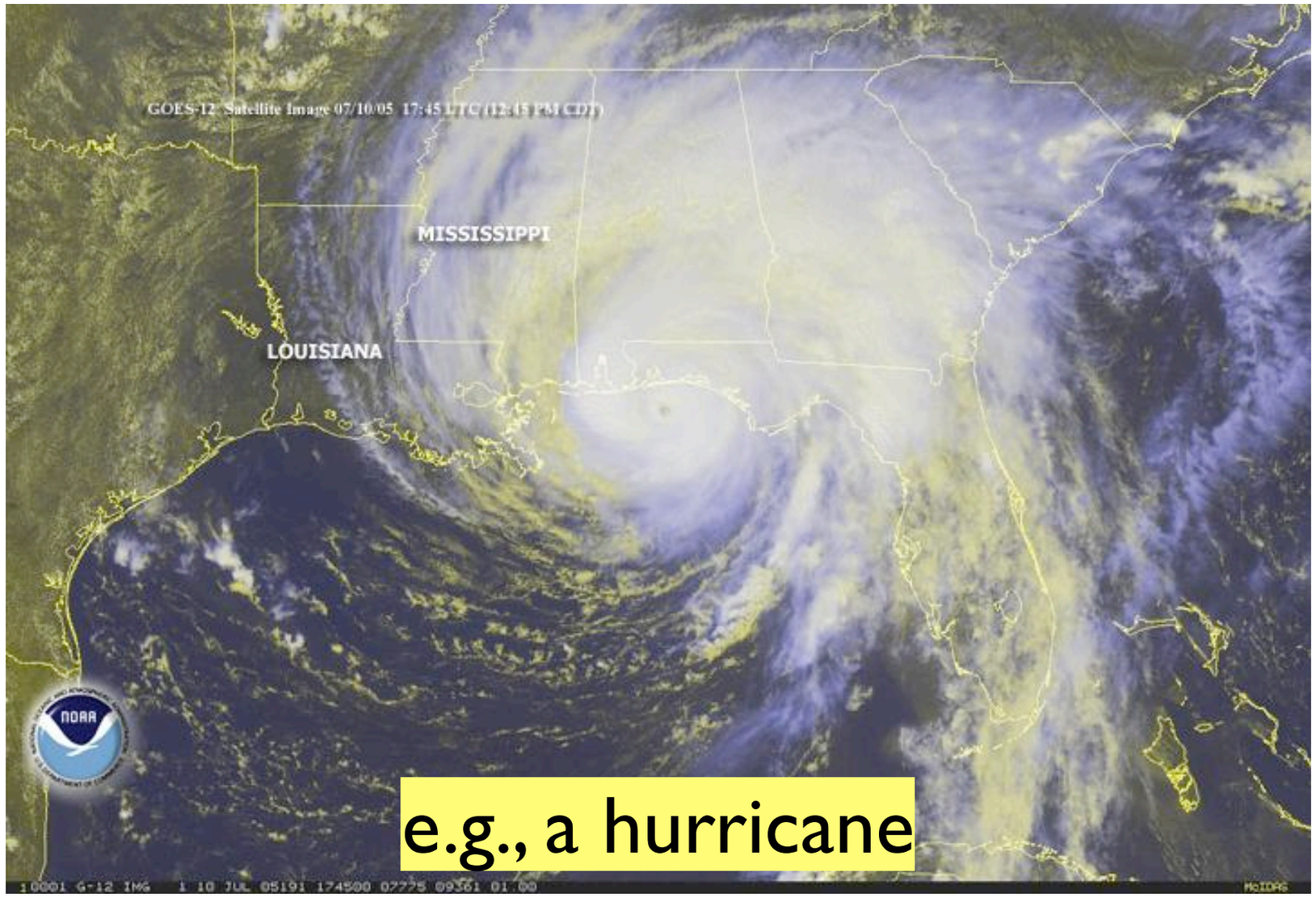
- Build system based on autoconf
- Coded in Fortran90, approximately 51 kLOC
- Distribution contains
  - mpeu -- Message Passing Environment Utilities
  - mct -- the toolkit itself
  - examples
    - simple -- single-source-file codes containing two components with sequential and concurrent component scheduling
    - climate_concur1 -- two-component example using concurrent scheduling
    - climate-sequen1 -- two-component example using sequential scheduling

Two Major MCT Applications

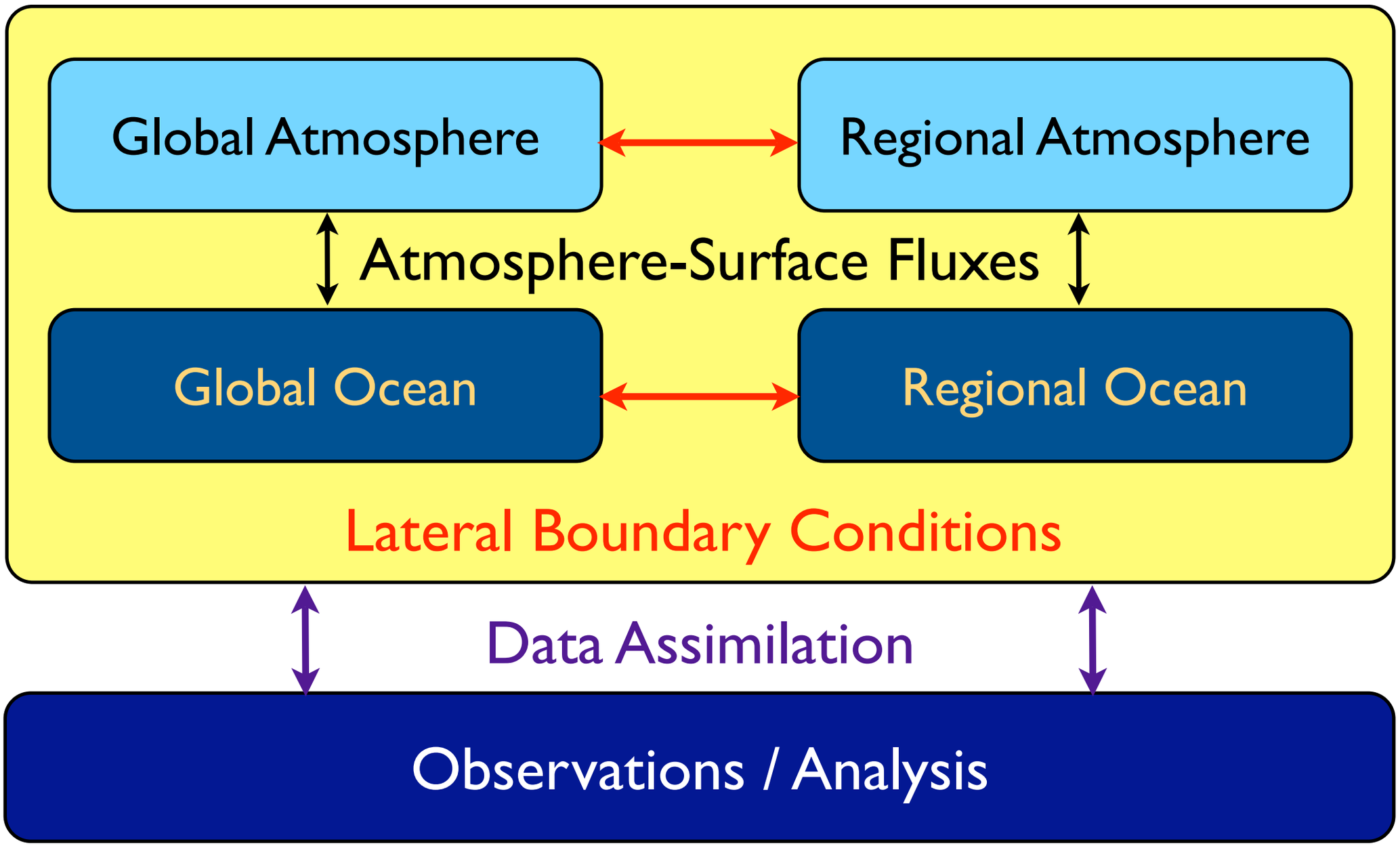
- Community Climate System Model CCSM 3.0
  - Flux coupler CPL6 -- Parallel "glue" to exchange data between parallel atmosphere, ocean, sea ice, and land models
  - Implemented as a toolkit that extends MCT
  - Used in ~11,000 model years' production runs and runs submitted to latest UN-sponsored Intergovernmental Panel on Climate Change assessment
- Weather Research and Forecasting (WRF) Model
  - Parallel Coupling API
  - Implemented as a parallel I/O mechanism
  - Available separately from WRF on MCT Web site
  - Has been run successfully in a computational grid environment

Q. What's in it for me?
A. The kind of modeling we really want to do...

...Is Multiphysics and/or Multiscale in Nature...



NWP for a Hurricane (if done all at once)



Complexity Barriers

Traditional approach: Model individual subsystems of a greater whole in isolation

- Idealize interactions with outside world using prescribed data or simplified physics
- Why? Three complexity barriers to overcome:
  - **Knowledge**, a consequence of specialization
    - overcome through interdisciplinary teams
  - **Computational**, i.e., getting all the math done
    - overcome by faster processors, better algorithms, and *parallel computing*
  - **Software**: build system, language barriers, interactions between physics packages

Message-passing parallelism creates a new software complexity barrier: **the parallel coupling problem** (PCP)

Parallel Coupling Problem

Given: N mutually interacting models (C1,C2,...CN), each of which may employ message-passing parallelism

Goal: Build an efficient parallel coupled model

Aspects of the problem:

- Architecture
- Parallel data processing
- Environment--important, beyond scope of current discussion
  - Language barriers
  - Build issues

Architectural Aspects

Two sources that shape parallel coupled models:

- Science of the system under study:
  - Connectivity--who talks to whom?
  - Domain overlap--lower-dimensional vs. colocation
  - Timescale separation/interaction & domain overlap
  - Coupling event scheduling (e.g., periodic?)
  - Tightness
- Implementation choices:
  - Resource allocation
  - Scheduling of model execution
  - Number of executable images
  - Mechanism

Dynamic Resource Allocation

Strategy	Intra-component	Inter-component	Global
Level 0 (Static)	Static	Static	Static
Level 1	Dynamic	Static	Static
Level 2	Dynamic	Dynamic	Static
Level 3	Dynamic	Dynamic	Dynamic

Requirements Stemming from Dynamic Load Balance

Cumulative growth in requirements as constraints on scheduling and resource allocation are loosened:

Level 0: No additional requirements

Level 1: Fast handshaking between components to cope with changing decompositions between coupling events

Level 2: Ability of framework or coupling mechanism to checkpoint models and re-instantiate/restart them on their new processor pools

Level 3: Ability of underlying communications mechanism to cope with dynamically varying global resource pool (e.g., dynamic MPI_COMM_WORLD)

Tightness vs. Looseness

A logical starting point is the cost break-down structure for the coupled model. Define the load matrix **L** as

$$L_{ij} = \begin{cases} \text{cost of running } C_i \text{ in isolation}, & \text{if } i = j, \\ \text{cost of delivering data from } C_j \text{ to } C_i, & \text{if } i \neq j. \end{cases}$$

The tightness of an individual model's coupling to the rest of the system is

$$\tau_i = \frac{1}{L_{ii}} \left(\sum_{j \neq i} L_{ij} + \sum_{j \neq i} L_{ji} \right).$$

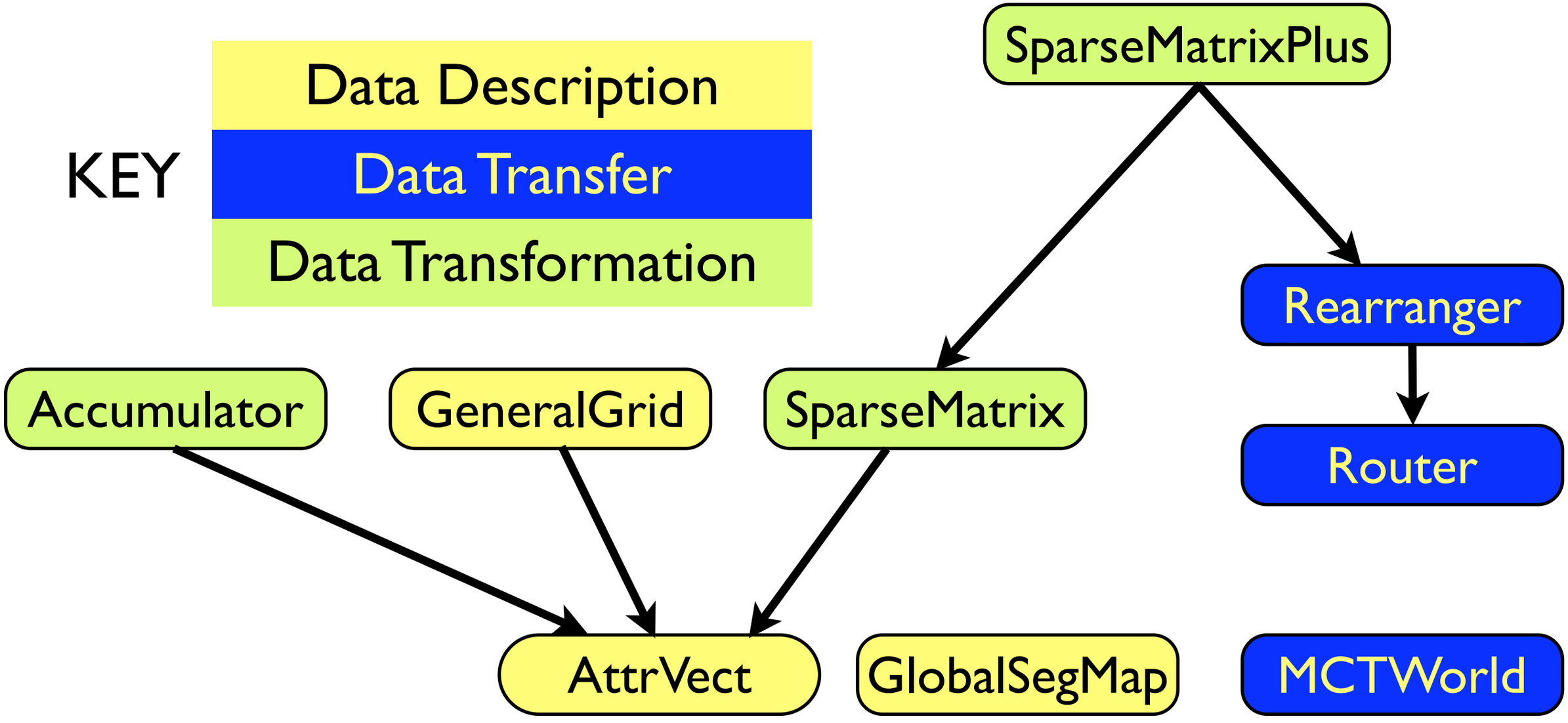
The overall tightness of a coupled system is the ratio of its coupling costs to total costs

$$T = \frac{\sum_{i \neq j} L_{ij}}{\left(\sum_{i \neq j} L_{ij} + \sum_{k=1}^{N} L_{kk} \right)}$$

Parallel Data Processing

- Description of data to be exchanged during coupling
  - Physical fields/variables
  - Mesh or representation associated with the data
  - Domain decomposition
- Transfer of data--a.k.a. the MxN problem
- Transformation of data
  - Intermesh interpolation/transformation between representations
  - Time transformation
  - Diagnostic/variable transformations
  - Merging of data from multiple sources

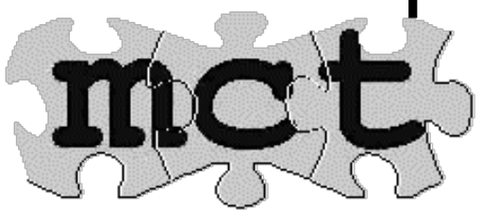
MCT is a Collection of Classes...



...where "class" is used following Decyk et al. (1996,1997)

MCT's Universe of Discourse

- Support coupling of **MPI-based MPP models**
- Data transfer using a **peer commmunication** model
- Description of physical meshes and associated field data through **linearization**
- Data transfer and transformation are viewed as multi-field, **pointwise** operations
- We leave numerous, high-level operations to the user's discretion (e.g., choice of linearization and interpolation schemes), while concentrating on automation of complex (but important!) low-level operations



Coupling as Peer Communication

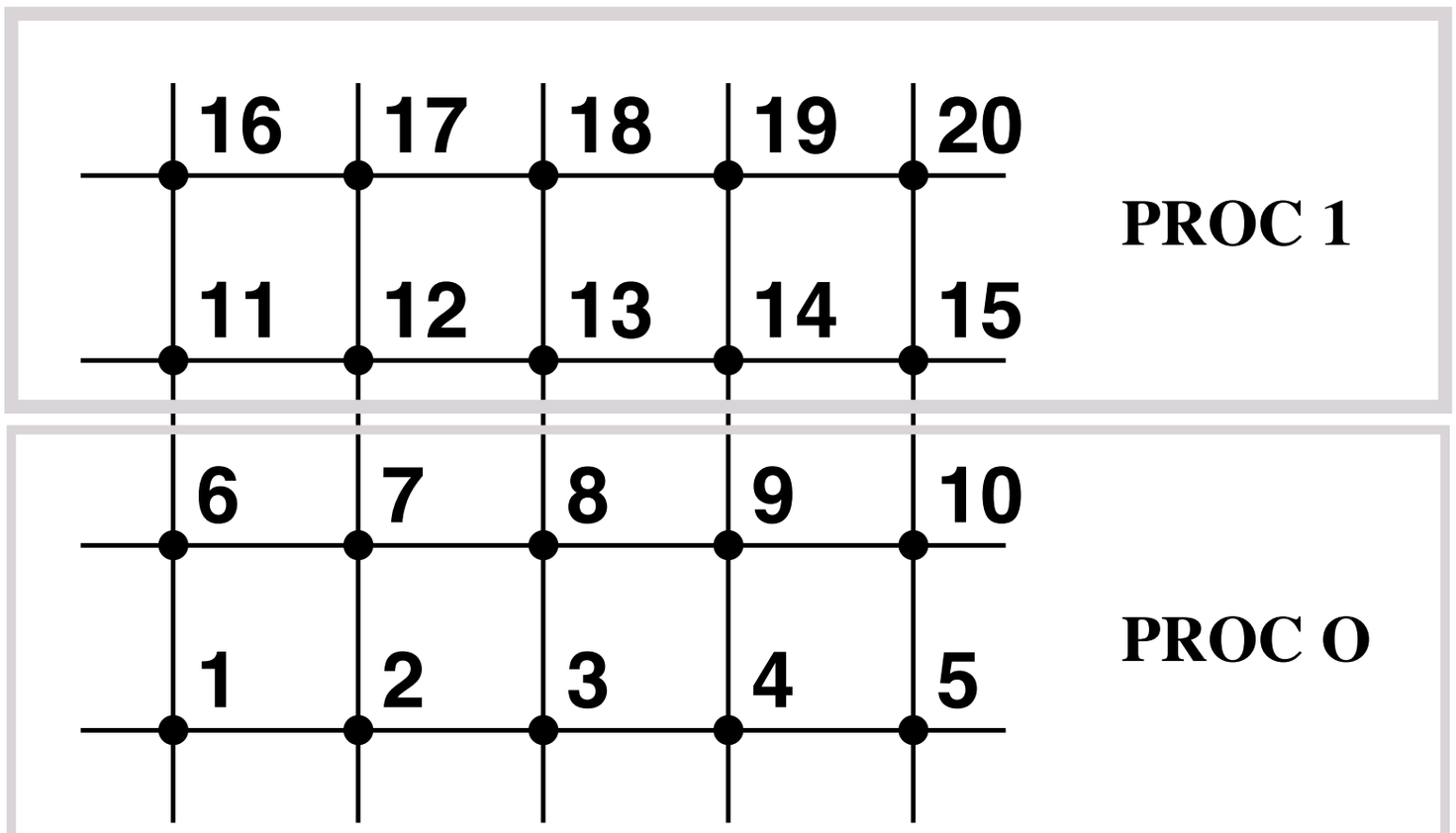
- MCT's organizing principle is the *component model*, or *component* (**not** same as CORBA, CCA, ESMF, JavaBeans)
- An MCT component is merely a model that is part of the larger system and participates in coupling
- In MCT, components interact directly as peers
- The user codes these connections into the model source

Linearization of Multi-Dimensional Space

- Linearization (first used in MxN schemes by UMD's METACHAOS) is the mapping from an n-tuple index space to a single global location index
- This approach allows for a single representation of grids/arrays of aribtrary dimension

A Simple Linearization Example

- 2D Cartesian mesh
- 2 Processors
- Numbering varies fastest in x-direction



Pointwise Operations

- FACT: Most coupling between components involves multivariate exchanges (e.g., ~10-12 fields between atmosphere and ocean in a climate model)
- Consolidating operations on data to be exchanged can result in performance boosts due to better cache re-use and lowered MPI latency charges
- In MCT, implementation choices (primarily storage order) have been made to exploit this situation

Automating (Most of) the Tough Stuff

- We leave coupled model architecture decisions to the user
- We aim to be minimally invasive--user still owns main(), still calls MPI_Init(), et cetera...
- The user decides how and when to couple
- The user must describe application grids and decompositions using MCT's GeneralGrid and GlobalSegMap classes
- The user must pack coupling data in AttrVect form
- MCT's library routines do the heavy lifting

Data Description: Domain Decomposition

- Embodied in the **GlobalSegMap** class
- Linearization creates a single unique index for each location, and runs of consecutive indices--or *segments*--are stored by start index, segment length, and processor ID on which it resides (on the component's communicator)
- Capable of supporting arbitrary decompositions and haloed decompositions
- Support for global-to-local and local-to-global index translation

Data Description: Physical Meshes

- Embodied in the **GeneralGrid** class
- Linearization implies this class stores real and integer attributes of individual mesh points
  - Coordinates, length, cross-sectional area, and volume elements, integer and real masks, grid indices
- Able to describe meshes of arbitrary dimensionality and also unstructured meshes
- Method support for sorting points in lexicographic order by coordinates and/or other attributes

Data Description: Field Data

- Embodied in MCT **AttrVect** class, which is similar to Trilinos multi-vector
- Allows storage of both integer and real attributes
- In implementation, major index is field index, keeping in line with pointwise approach
- Attributes are accessed via use of string tokens
- Lists of attributes are set at initialization, which allows run-time binding of what fields are stored
- Numerous query and manipulation methods including importing and exporting attributes and MergeSort keyed by attributes

Data Transfer: Registration

- Singleton class **MCTWorld** holds a lightweight component registry
- Components are registered with a component ID
- MCTWorld contains a rank translation table that allows a component to message another remote component by using information from its local domain decomposition descriptor (i.e., we do not construct nor use intercommunicators)

Data Transfer: Communications Scheduling

- Scheduling for one-way parallel data transfers are handled by the **Router** class
- Scheduling for two-way parallel data transfers and parallel data redistributions are handled by the **Rearranger** class, which comprises two Routers

Data Transfer: Execution

- MxN Communications between components are carried out by library routines MCT_Send() and MCT_Recv(), both of which have (overall) blocking and non-blocking versions
  - Overall blocking--Prevents concurrently scheduled components from outrunning each other
  - Overall nonblocking--allows for same communications approach to be used for sequentially scheduled components
    - N.B.: non-blocking MPI operations are used for the individual point-to-point messages within the MxN transfer operation
- MxM redistributions are handled by the MCT library routine Rearrange()
  - Contains logic to prevent self-messaging (does a copy instead)

Data Transformation: Intermesh Interpolation

- MCT's linearization-based worldview casts interpolation as a linear transform and thus implementable as a matrix-vector multiply
- In practice, these interpolation matrices are quite sparse
- MCT's **SparseMatrix** class provides storage for nonzero matrix elements in COO format
- Method support includes query methods, computation of sparsity, sorting methods to aid matrix decomposition and to boost performance
- Library function performs multiplication **y** = **Mx**, where **x** and **y** are of AttrVect type, and this function performs automatic token-based matching of attributes
- Main implementation targets commodity cache-based processor platforms, and MCT's pointwise operation view makes good re-use of cache
- Modifications for vector platforms also included

3 Ways to Parallelize **y = Mx**

Based on **x**
(**y** is shorter than **x**)

1) Use local data to form (embarassingly parallel) partial sums **y'**

y' = Mx

2) Communicate to reduce partial sums **y'** to final product **y**

Based on **y**
(**x** is shorter than **y**)

1) Gather pre-image of **y**
x' from **x**

2) Compute the product

y = Mx'

(embarassingly parallel)

Other
(e.g. Based on **M**)

1) Gather pre-image of **y**
x' from **x**

2) Compute the partial sums

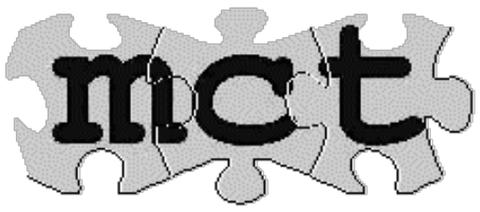
y' = Mx'

(embarassingly parallel)

3) Communicate to reduce partial sums **y'** to final product **y**

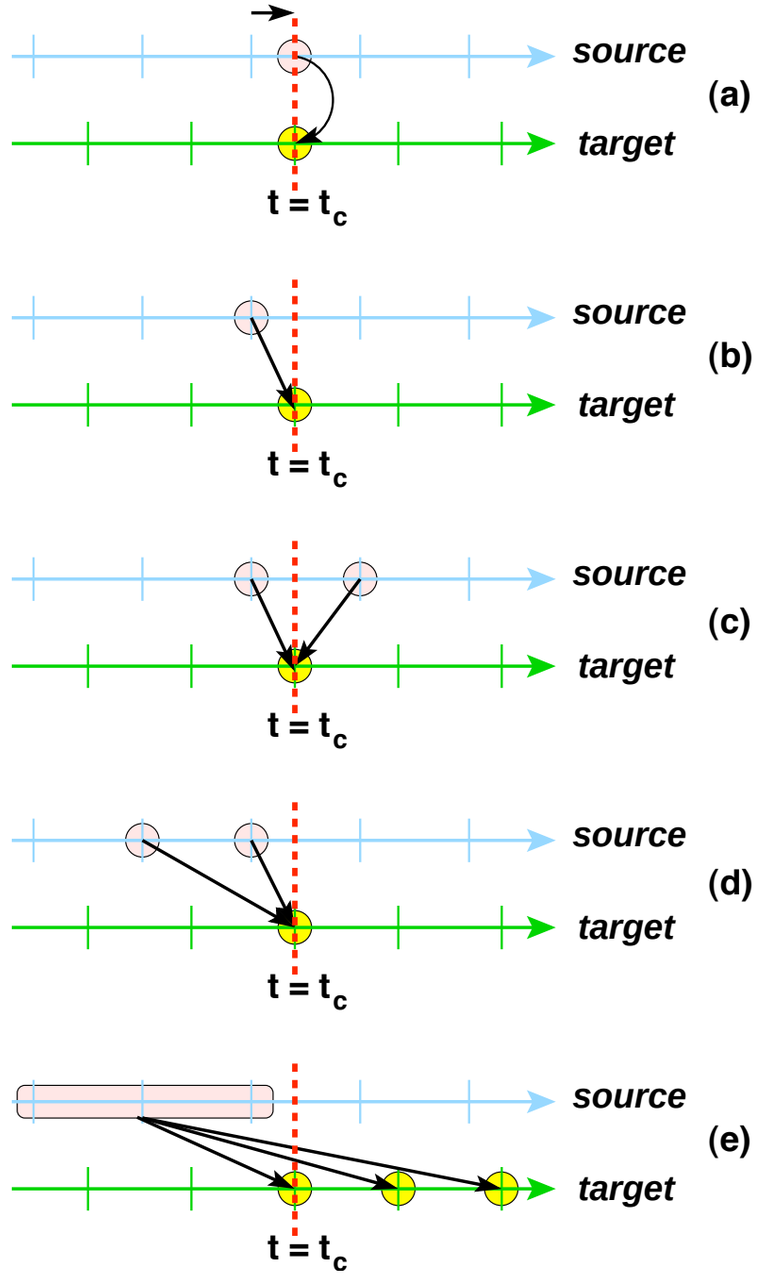
MCT's Parallel Linear Transformation

- MCT provides the **SparseMatrixPlus** class, which encapsulates everything needed for a sparse linear transform
- This class includes Rearrangers to schedule communications and a SparseMatrix to store matrix elements
- Library routine to compute **y** = **Mx** in parallel, again where **x** and **y** are of AttrVect type, and automatic token-based matching of attributes is performed
- Support for both commodity and vector processors



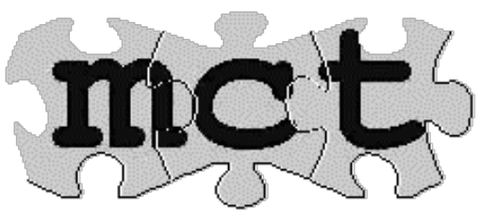
Data Transformation: Time Accumulation

- Time evolution of multi-component systems can involve data exchanges of *instantaneous* or/or *integrated* data
- For instantaneous exchanges, use of one or more AttrVects is sufficient
- For integrated data exchanges, MCT offers accumulation registers in the form of the **Accumulator** class, and the accumulate() library routine
- The Accumulator provides registers for time integration and averaging of data, and keeps track of progress over an accumulation cycle
- The accumulate() library routine works with AttrVect and Accumulator arguments, and automatically cross-indexes and accumulates attributes with matching tokens



Other Services--Spatial Integration and Averaging

- MCT provides routines for computing spatial integrals and averages
- These routines are included to diagnose (and if wished, even enforce) conservation of integrals in the interpolation process
- Library routines operate on AttrVect objects
- Spatial weight elements and masks can be provided either in GeneralGrid or array form
- Paired integral/average routines to compute integrals simultaneously on source and target grids to minimize global sum latency costs



Other Services--Merging of Data from Multiple Sources

- Often one must combine outputs from multiple components for use as input for another component (e.g., fluxes/states from ocean, land, and sea-ice for use by atmosphere)
- MCT provides a **Merge** facility, which is a set of routines to combine multiple AttrVect data streams into a single AttrVect result
- Real and Integer masks for the merge can be supplied either in GeneralGrid or array form
- Automatic token-based attribute matching

MCT Programming Model

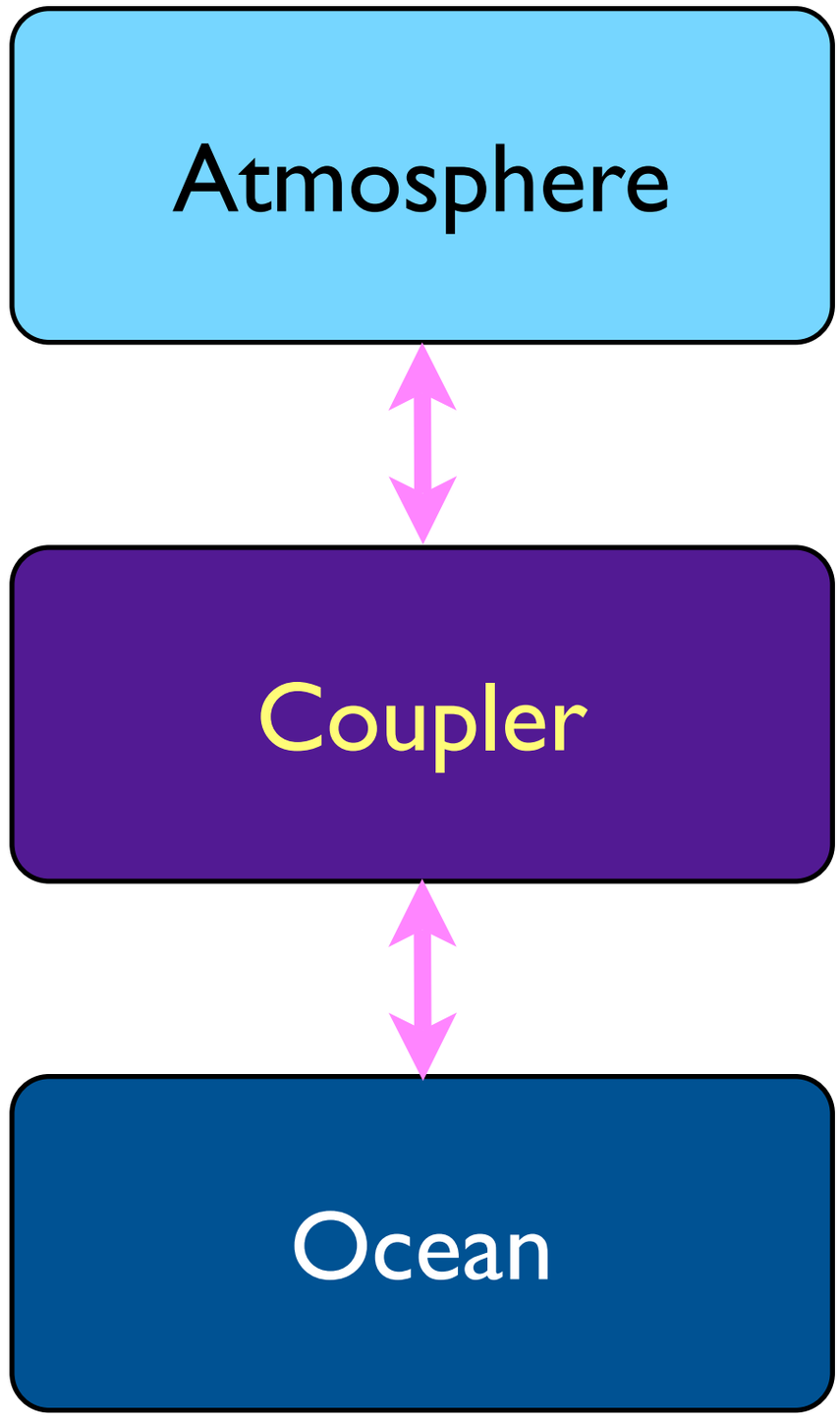
Based on Fortran90, but we are beginning to support other languages.

- **Use** modules for access to MCT classes and methods
- **Declare** variables of MCT datatypes
- **Invoke** MCT library routines to accomplish parallel coupling operations

A Simple Example, Where Simple is a Relative Term

A (Sort of) Simple Example

- Three Components
- Coupler component handles data transformation
  - Interpolation
  - Time averaging
- Components communicate with coupler
  - Atmosphere hourly
  - Ocean daily



Further Details and Simplifying Assumptions

- Atmosphere and Ocean employ simple 2D latitude/longitude grids at surface
  - Atmosphere: aLats x aLons = NatmPts
  - Ocean: oLats x oLons = NocnPts
- Coupler uses the same decomposition for Atmosphere for all of its processing
- Coupler uses the same decomposition for the Ocean for all of its processing
- Atmosphere outputs same fields that coupler receives as input and vice-versa
- Ocean outputs same fields that coupler receives as input and vice-versa

Module Use (ATM, OCN)

```
program ATMorOCN
.
.
.
  use m_MCTWorld
  use m_GeneralGrid
  use m_GlobalSegMap
  use m_AttrVect
  use m_Router
  use m_Transfer
  use m_Accumulator
.
  implicit none
.
```

Component Registry

Physical Mesh Descriptor

Domain Decomposition Descriptor

Exchanged Field Data Storage

MxN Comms Scheduler

MCT MxN Transfer Routines

Time Integration Registers (Ocean Only)

Module Use (Coupler)

```
program Coupler
.
.
.
  use m_MCTWorld
  use m_GeneralGrid
  use m_GlobalSegMap
  use m_AttrVect
  use m_Router
  use m_Transfer
  use m_SparseMatrix
  use m_SparseMatrixPlus
  use m_MatAttrVectMul
  use m_Accumulator
.
.
  implicit none
.
```

Component Registry

Physical Mesh Descriptor

Domain Decomposition Descriptor

Exchanged Field Data Storage

MxN Comms Scheduler

MCT MxN Transfer Routines

Interpolation Weight Storage

Parallel Interpolation Matrix Object

Parallel Matrix-Attribute Vector Multiply

Time Averaging/Integration Registers

Declaration (Atmosphere)

```
! Domain decomposition descriptor for Atmosphere
  Type(GlobalSegMap) :: AtmDecomp
! MCT description of Atmosphere grid
  Type(GeneralGrid)  :: AtmMesh
! MCT storage for outgoing/incoming field data
  Type(AttrVect)  :: Atm2Cpl, Cpl2Atm
! MCT Comms scheduler for MxN to/from Coupler
  Type(Router)  :: AtmCplSchedule
```

Declaration (Ocean)

```
! Domain decomposition descriptor for Ocean
  Type(GlobalSegMap) :: OcnDecomp
! MCT description of Ocean grid
  Type(GeneralGrid)  :: OcnMesh
! MCT storage for incoming/outgoing field data
  Type(AttrVect)  :: OcnFromCpl, OcnToCplInst
! MCT Comms scheduler for MxN to/from Coupler
  Type(Router)  :: OcnCplSchedule
! Time Averaging/Integration registers for output
  Type(Accumulator)  :: OcnToCplAccum
```

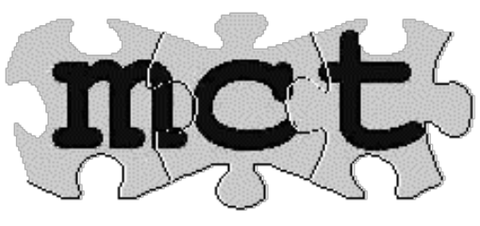
Declaration (Coupler)

```
! Domain decomposition descriptors for Atm/Ocn
  Type(GlobalSegMap) :: AtmCplDecomp, OcnCplDecomp
! MCT description of Atmosphere and Ocean grids
  Type(GeneralGrid)  :: AtmMesh, OcnMesh
! MCT storage for field data to be processed
  Type(AttrVect)  :: CplToAtm, CplToOcn, CplFromOcn, CplFromAtm
! MCT comms scheduler for MxN to/from Coupler
  Type(Router)  :: CplAtmSchedule, CplOcnSchedule
! Time averaging/integration registers for output
  Type(Accumulator)  :: AtmToOcn
! Storage for interpolation matrix elements
  Type(SparseMatrix)  :: A2OMatElements, O2AMatElements
! Parallel interpolation matrix-AttrVect multiply objects
  Type(SparseMatrixPlus)  :: A2OparXform, O2AparXform
```

Invocation: MCT Initialization

- Purely concurrent component scheduling--each component on its own pool of processors
- Shown for Atmosphere, but is representative
- MCT component IDs: Atmosphere=1, Ocean=2, Coupler=3
- Works for either single or multiple executable application

```
...
! MPI initialization
  call MPI_INIT(MPI_COMM_WORLD, ierr)
! Split MPI_COMM_WORLD to get atmosphere communicator AtmComm
...
  compID = 1
! Initialize MCT World Registry
  call MCTWorld_init(1,MPI_COMM_WORLD, AtmComm, compID)
```



Invocation: MCT Initialization

- Purely sequential component scheduling--each component on its own communicator, which is a copy of MPI_COMM_WORLD
- MCT component IDs: Atmosphere=1, Ocean=2, Coupler=3
- Shown for driver that calls Atmosphere, Ocean, and Coupler

```
! MPI initialization
  call MPI_INIT(MPI_COMM_WORLD, ierr)
! Copy MPI_COMM_WORLD to get atmosphere communicator AtmComm,
! ocean communicator OcnComm, and coupler communicator CplComm
...
! Set component IDs
  compIDs(1) = 1 ! Atmosphere
  compIDs(2) = 2 ! Ocean
  compIDs(3) = 3 ! Coupler
! Initialize MCT World Registry
  call MCTWorld_init(3,MPI_COMM_WORLD, AtmComm, compIDs)
```

GlobalSegMap Initialization

- Shown for Atmosphere, but representative
- Arguments
  - starts(:) - Integer array of starting indices (global) for each segment
  - lengths(:) - Integer array of segment lengths
  - root - Integer, rank of root process on component's local communicator AtmComm
  - AtmCompID - Integer, MCT component ID number

```
! Initialize Atmosphere GlobalSegMap
  call GlobalSegMap_init(AtmDecomp, starts, lengths, root, &
                                    AtmComm, AtmCompID)
```

AttrVect Initialization

- Shown for Ocean output to Coupler, but representative
- Attributes (second and third string arguments, respectively):
  - Integer: Latitude, Longitude, and Linearized indices (all global)
  - Real: Temperature, east component of current, north component of current, EW gradient of surface height, NS gradient of surface height, and heat flux
    - nOcnLocal - Number of points on this processor's chunk of ocean grid
    - N.B.: This call *constructs* the AttrVect; data is filled in separately

```
! Initialize Ocean Output AttrVect Ocn2CplInst
  call AttrVect_init(Ocn2CplInst, 'LatInd:LonInd:LinInd', &
                                  'T:u:v:S:dhdx:dhdy:Q', nOcnLocal)
```

Router Initialization for Atmosphere/Coupler MxN

In the Atmosphere:

```
! Initialize Atmosphere-to-Coupler Router
  call Router_init(CplCompID, AtmDecomp, AtmComm, &
                   AtmCplSchedule)
```

Router

In the Coupler:

```
! Initialize Coupler-to-Atmosphere Router
  call Router_init(AtmCompID, AtmCplDecomp, CplComm, &
                   CplAtmSchedule)
```

Router

SparseMatrix Initialization

- MCT does *not* currently generate interpolation weights on-line
- Instead, we rely on other tools to generate them off-line, or at start-up
- Suppose we have a sparse interpolation matrix **M** whose elements are specified in COO format by three arrays:
  - grows(:) - integer, global row indices
  - gcols(:) - integer global column indices
  - weights(:) - real matrix entries
- For this example, we are calling this routine only on the coupler's root process; lsize is the number of nonzero elements

```
call SparseMatrix_init(A2OMatElements, nrows, ncols, lsize)
call SparseMatrix_importGlobalRowIndices(A2OMatElements, grows, lsize)
call SparsMatrix_importGlobalColumnIndices(A2OMatElements, gcols, lsize)
call SparseMatrix_importMatrixElements(A2OMatElements, weights, lsize)
```

SparseMatrixPlus Initialization

- Shown for Coupler's Atmosphere-to-Ocean transformation
- Uses SparseMatrix object A2OMatElements created on previous slide
- MvMultDecompStrategy is the parallelization strategy choice, which is one of
  - row-based
  - column-based
  - matrix-based (e.g., use graph partitioning for load balancing the multiply)
- N.B.: This call *constructs and fills in* the SparseMatrixPlus

```
call SparseMatrixPlus_init(A2OParXform, A20MatElements, &
                           AtmCplDecomp, OcnCplDecomp, &
                           MvMultDecompStrategy, CplRoot, &
                           CplComm, CplCompID, Tag)
```

Accumulator Initialization

- Shown for Ocean output to Coupler, but representative
- Only real attributes are being accumulated
- rAction - integer array of processing action MCT_SUM (0, summation) or MCT_AVG (1, average assuming uniform time weighting)
  - For this example, rAction = (1 1 1 1 1 1 0)
- nOcnLocal - Number of points on this processor's chunk of ocean grid
- num_steps - number of steps in an accumulation period (1 day with 20 minute ocean timestep)
- steps_done - set to zero if we are beginning at the start of an accumulation period
- N.B.: This call *constructs* the Accumulator; data is filled in separately

```
! Initialize Ocean Output Accumulator Ocn2CplAccum
  call Accumulator_init(Ocn2CplAccum, rList='T:u:v:S:dhdx:dhdy:Q', &
                        rAction=rActions, lsize=nOcnLocal, &
                        num_steps=72, steps_done=0)
```

MxN Transfer ATM to CPL (Blocking)

In the Atmosphere:

```
! Send Attributes in Atm2Cpl to Coupler
  call MCT_Send(Atm2Cpl, AtmCplSchedule, Tag)
```

In the Coupler:

```
! Receive Attributes in CplFromAtm from Atmosphere
  call MCT_Recv(CplFromAtm, CplAtmSchedule, Tag)
```

MxN Transfer ATM to CPL (Non-Blocking)

In the Atmosphere:

```
! Send Attributes in Atm2Cpl to Coupler
  call MCT_ISend(Atm2Cpl, AtmCplSchedule, Tag)
...
  call MCT_WaitSend(AtmCplSchedule)
```

In the Coupler:

```
! Receive Attributes in CplFromAtm from Atmosphere
  call MCT_IRecv(CplFromAtm, CplAtmSchedule, Tag)
...
  call MCT_WaitRecv(CplAtmSchedule)
```

Parallel Interpolation in the Coupler (Atm to Ocn)

- Message-passing parallel matrix-vector multiply to interpolate Atmosphere real fields onto Ocean real fields
- As usual in MCT, automatic token-based attribute matching coordinates the calculation

```
! Perform parallel sparse matrix-attribute vector multiply:
  call sMatAvMult(CplFromAtm, A2OParXform, CplToOcn)
```

Accumulation in Ocean

- Time averaging of states and summation of fluxes in the Ocean
  - Performed every ocean timestep
  - Accumulation period one model day, which matches the coupling period for this component
    - As usual in MCT, automatic token-based attribute matching guides the calculation

```
! Accumulate instantaneous ocean data in AttrVect OcnToCplInst into
! the accumulator OcnToCplAccum
  call accumulate(OcnToCplInst, OcnToCplAccum)
```

OK, So You Hate Fortran

- Work has begun to export MCT's programming model to other languages
- Accomplished by leveraging CCA's Babel language interoperability tool
  - SIDL description of a *restricted version* of the MCT API
  - Bindings for this API generated automatically for both C++ and Python
  - Download ftp://mcs.anl.gov/pub/acpi/MCT/babel.tar.gz
  - Foundation code for pyMCT and pyCPL (U. Chicago)

Conclusions

- MCT exists, is highly portable and robust
- MCT supports a linearization approach to coupling that is universally applicable to mesh-based systems
- MCT services support numerous modes of operation for coupled systems
  - Purely sequential component scheduling
  - Purely concurrent component scheduling (including multiple executables and computational grids)
  - Combinations thereof

Future Work

- Continue expanding the programming model to other languages using Babel
- Address explicitly one-dimensional approach and craft more convenient interfaces for multidimensional grids and arrays
- Speed up intercomponent handshaking to better support level intracomponent dynamic load balance
- Modify registry to support level intercomponent dynamic load balance (but somebody else can do the checkpointing)
- Include support for pure OpenMP and hybrid parallelism
- Move beyond linear transformations
- More sophisticated Merge facility

DOE ACTS Workshop, Berkeley, CA, August 23-26, 2005